University of Bahrain
College of Information Technology
Department of Computer Science
First Semester, 2015-2016

ITCS215 (Data Structures)

Test I

Date: 21/10/2015 Time: 16:00 - 17:15

STUDENT NAME (Uppercase characters)

STUDENT ID#	**2**	**0**						

SECTION#	

NOTE: THERE ARE SIX **(6) PAGES** IN THIS TEST

ONLY ONE SOLUTION WILL BE CONSIDERED FOR EACH QUESTION

QUESTION#	MARKS		COMMENTS
1	12		
2	16		
3	12		
TOTAL	40		

Question 1 [6+6 Marks] [Inheritance]

Consider the following class definition:

```
class Account
{
private:
     string       userName;
     int          accountID;
     float        balance;
public:
     account(string name, int ID, float bal);
     string getUserName();
     int getAccountID();
     float getBalance();
     void setBalance(float bal);
     void print(); // prints all attributes
};
```

(A) Write a class called **InterestAccount** which inherits all the properties of class **Account** with inheritance type as public. This new class will have the following additional members:

Data member:

interestRate (float)

Member functions:

1. Set function for data member interestRate.
2. Get function for data member interestRate.
3. Constructor having 4 parameters, for all attributes (including that of class Account).
4. Function **addInterest** that updates the balance using the following equation:
   `balance = balance + (balance * interestRate / 100) / 12.`
5. Function **print** to print all attributes (including that of **Account**).

Write only prototype of all member functions int the class **InterestAccount.**

Solution

```
class InterestAccount : public Account
{

private:
   float interestRate;

public:
   void SetInterest(float i);
   float GetInterest();
   InterestAccount(string u, int id, float b, float i);
   void addInterest();
   void print();
};
```

(B) Write definitions (implementation) of the following member functions of class **InterestAccount:**

constructor, **addInterest** and **print**.

constructor:

```
InterestAccount::InterestAccount(string u, int id, float b, float i)
         :Account(u,id,b)
         {
          interestRate=i;
         }
```

addInterest:

```
void InterestAccount::addInterest()
{
 float b;
 b=(getBalance()+((getBalance()* interestRate / 100))/12);
 setBalance(b);
}
```

print:

```
void InterestAccount::print()
{
 Account::print();
 cout<<"Interest Rate: "<<interestRate<<endl;
}
```

Question 2 [8+8 Marks] [Array Based List]

(A) [8 Marks] Write a member function **removeMax** to be included in class **arrayListType** having a reference parameter **max** of type **elemType**. the function finds the maximum element of the list and assign it to parameter **max** and also deletes all occurrences of **max** from the list. If the list is empty, the function returns false, else the function returns true after deletion.

Function prototype:

```
     bool removeMax(elemType& max);
```

Solution:

```
template <class elemType>
bool arrayListType<elemType>::removeMax(elemType &max)
{
  if(isEmpty())
   {
    count<<"List is empty"<<endl;
    return false;
   }

  int max_loc=0;
  for(int i=1; i < length; i++)
   if(list[i] > list[max_loc])
   {
     max_loc=i;
     max=list[i];
   }
  for(int j=0; j < length; j++)
   if(list[j]==max)
     {
      removeAt(j);
      j--;
     }

   return true;
}
```

(B) **[8 Marks]** Wite a non-member function **replaceSides** that has two parameters, **item** of type **elemType** and **list1** of type **arrayListType**. The function compares the first element of **list1** with the last element of **list1**. If they are equal, it replaces both of them by **item**. Similarly, the same process will be done for the second element of **list1** and the element before the last and so on for the rest of the elements. If any replacement has been done, the function returns true, otherwise the function returns false. If **list1** is empty, the function also returns false.
Assume that the number of elements in **list1** is even.

Function prototype:

```
template <class elemType>
bool replaceSides(const elemType& item, arrayListType<elemType>& list1);
```

Example:

Assume **item=40**

list1 (Before function call):	20	15	17	8	70	17	78	20
list1 (After function call):	**40**	15	**40**	8	70	**40**	78	**40**

The function return **true**.

Solution:

```
template <class elemType>
bool replaceSides(const elemType&item, arrayListType<elemType>&list1)
{
  if (list1.isEmpty())
    return false;

  bool found=false;
  elemType x1,x2;
  int last=list1.listSize()-1;
  int mid=list1.listSize()/2;
  for(int i=0; i < mid; i++)
  {
    list1.retrieveAt(i,x1);
    list1.retrieveAt(last,x2);
  if(x1==x2)
   {
     list1.replaceAt(i,item);
     list1.replaceAt(last,item);
     found=true;
   }

  last--;
  }

  return found;
}
```

Question 3 [12 Marks] [Linked List]

Write a member function called **insertSpecial** to be included in class **linkedListType** to insert a data **newItem** in linked list as follows:

(a) If the list is empty, do not insert any node and return false.

(b) If the **insertSpecial** is less than the **info** of the first node, then insert a new node having **newItem** of the node, at the begining of the list and return true.

(c) If the **newItem** is greater than the **info** of the last node, the insert a new node having **newItem** as **info** of the node, at the end of the list and return true.

(d) In all other cases, do not insert any node and return false.

Function Prototype:

```
bool insertSpecial(const Type& newItem);
```

Do not call any member function of class linkedListType in your member function.

Solution:

```
template <class Type>
bool linkedListType<Type>::insertSpecial(const Type& newItem)
{
  if (first==NULL)
    {
    cout<<"The list is empty"<<endl;
    return false;
    }
  nodeType<Type> *newNode;
  newNode=new nodeType<Type>;
  assert(newNode!=NULL);
  newNode->info=newItem;
  if(newNode->info < first->info)
    {
    newNode->link=first;
    first=newNode;
    count++;
    return true;
    }
  else if(newNode->info > last->info)
    {
    newNode->link=NULL;
    last->link=newNode;
    count++;
    return true;
    }
  else
    return false;
}
```